



PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
DEPARTAMENTO DE CIENCIA DE LA COMPUTACIÓN
IIC2343 - ARQUITECTURA DE COMPUTADORES

Aspectos Formales Tareas

1º semestre 2019 - **Profesor:** Yadran Eterovic

1. Introducción

IMPORTANTE: Todo aspecto mencionado en el siguiente documento respecto a las tareas del ramo Arquitectura de Computadores (IIC2343) se asume como aceptado por parte del alumno.

1.1. Tipos de Tareas

Las tareas de éste curso se dividen en dos tipos:

1. Programadas
2. Placa

1.2. Sobre el formato de entrega de las tareas

Más adelante en este documento especificaremos que significa cada una de ellas. Lo importante de esto es que ambos formatos de tarea deben ser entregados a través de la plataforma [GitHub](#) en los repositorios personales respectivos creados por el equipo de ayudantes. Es altamente recomendado el manejo de [Git](#)

Cada tarea deberá encontrarse en un repositorio **Tarea_XX_usuario**. Donde *usuario* corresponde, a su usuario Github y XX el numero de la tarea correspondiente (01, 02, 03, etc...).

Además se efectuará un descuento por tiempo de atraso, el cual corresponde a **1.0** menos en la nota final de la tarea **por cada 6 horas, o fracción** de atraso.

1.3. Sobre las solicitudes de corrección de tareas

Tendrán un plazo de una semana posterior a la entrega de una nota (con su respectivo feedback) para solicitar una corrección de la tarea. El proceso siempre será el mismo: se habilitará un google form donde deberán argumentar correctamente por que se debe corregir y que partes específicamente.

IMPORTANTE:

1. Esta permitido que arreglen lineas de su código para la corrección, en tal caso, deberán indicar el commit (ver [anexo git](#) en caso de no saber que es) que quieren que sera revisado. Además, se descontará una décima (0.1) de su nota final en la tarea por cada linea cambiada en el commit.
2. **NO SE DARÁ MAS PLAZO** para la entrega de tareas, sin excepción.

1.4. Archivo README.md

Toda tarea **debe** incluir un archivo README en formato [MarkDown \(.md\)](#), éste debe incluir:

1. Descripción breve de cómo funciona su programa/tarea
2. Objetivos logrados, parcialmente logrados y no logrados de su tarea
3. En caso que sea pertinente, respuestas a preguntas teóricas que puedan haber sido formuladas en el enunciado de la tarea

Importante: Tendrán 24 hrs a partir de la hora de entrega para subir su archivo README.md

1.5. Dudas respecto a tareas (enunciados)

Cualquier duda respecto a el contenido de una tarea en específico debe ser realizada a través de una Issue en el Syllabus (GitHub) del curso en el siguiente formato:

[TXX YYYY] - Título descriptivo de la duda

Dónde XX corresponde al numero de tarea, YYYY el tipo de la tarea (PROG o PLAC), haciendo referencia a si la tarea es programada o se necesita el uso de placa. Hacemos énfasis en que el título sea descriptivo (pero corto), de manera que sus compañeros puedan llegar fácilmente a esa *issue* en caso de tener una duda similar.

Ejemplo:

[T03 PROG] - ¿Que és Python?

Además, para no colapsar las *issue*, antes de hacer una pregunta verifiquen que ésta no haya sido realizada antes por algún compañero. Ideal también que si una duda sigue la misma línea de otra que ya esta respondida, se pregunte en un comentario de la *issue* existente.

En la sección *issue* podrán encontrar las **reglas del foro** donde se explica más del tema.

2. Tareas Programadas

Las tareas programadas son *estrictamente individuales*. Cualquier tipo de falta a el **Código de Honor** de la universidad será sancionada con la **reprobación** del curso de manera inmediata y con la nota mínima.

Estas deben ser desarrolladas en **Python 3.6**, y el archivo .py a ejecutar se debe llamar como se especifique en cada enunciado. Pueden crear cuantos módulos/archivos crean necesarios, pero solo se correrá el archivo correspondiente.

Su código debe guiarse por el formato **PEP8**, **habrá descuento en caso contrario** (Máx 5 décimas a criterio del corrector).

2.1. Sobre uso de librerías

Toda librería de la *librería estándar* de Python 3.6 está permitida, en caso de querer usar una librería externa, deben consultar a través de una Issue en GitHub específica para cada tarea.

2.2. Sobre formato de ejecución

Sus archivos deben ser ejecutables por consola y aceptar argumentos en el siguiente formato (en caso de ser necesario):

Supongamos un programa que necesita de un archivo de input y output, este debe lograr ejecutarse de la siguiente forma:

```
python program.py input_path.txt output_path.txt
```

En caso de no saber como lograrlo, ver **anexo**

3. Tareas en Placa

Las tareas en placa son en parejas designadas por el equipo de ayudantes, y solo entre los miembros de las parejas. Cualquier tipo de falta a el **Código de Honor** de la universidad será sancionada con la **reprobación** del curso de manera inmediata y con la nota mínima. Estas son desarrolladas en [Vivado](#), y es escrita en el lenguaje descriptivo de circuitos [VHDL](#).

En el repositorio oficial del curso encontrarán tutoriales de cómo descargar e instalar los programas necesarios para utilizar la placa.

3.1. Formato entrega tareas placa

El repositorio debe contener una carpeta con su proyecto de Vivado y el archivo .bit. En el caso de la carpeta del proyecto, deben subir solo la carpeta basic computer.srscs y el archivo basic computer.xpr. Considere que esta parte incluirá una evaluación de pares, que será detallada durante la entrega.

Dado que Vivado y VHDL es (o debería ser) desconocido para la mayoría, se realizaran talleres especiales donde se hará una pequeña introducción a ambos conceptos (Las fechas serán avisadas a su debido tiempo). Es altamente recomendado asistir a estas instancias.

4. Anexos

4.1. Git

Git es un sistema distribuido de control de versión, gratuito y open source, diseñado para manejar de pequeños a enormes proyectos de forma rápida y eficiente.

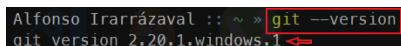
En otras palabras, permite un manejo de distintas versiones de un proyecto, manteniendo registro de las distintas etapas que ha tenido éste. Cada una de éstas etapas deben ser demarcadas manualmente, y se les llama **commits** (algo así como *checkpoints*). Gracias a esto entonces, podemos ver como ha ido evolucionando un proyecto, y a la vez tenemos la posibilidad de volver a versiones (commits) anteriores de nuestro proyecto en caso de que se necesite revisar/empezar de nuevo cierta parte.

Además, facilita el trabajo en equipo, permitiendo que dos personas trabajen en un mismo proyecto, sin editar directamente el código que está realizando la otra persona (en este curso no necesitaran hacer esto, pero vale la pena mencionarlo).

Algunas ventajas de el uso de Git son:

1. Trabajo en equipo fluido
2. Versiones disponibles en cualquier momento
3. Control de cambios efectuados a lo largo del proyecto
4. Fácil programación en paralelo, y posterior unión (**merge**) de ambas partes
5. Múltiples *backups* del proyecto

Git se basa en la linea de comandos (terminal, cmd, etc...) para realizar sus instrucciones. Primero que nada, deben tener [Git instalado](#) en sus computadores, pueden revisar si esta instalado corriendo la siguiente instrucción de su terminal:



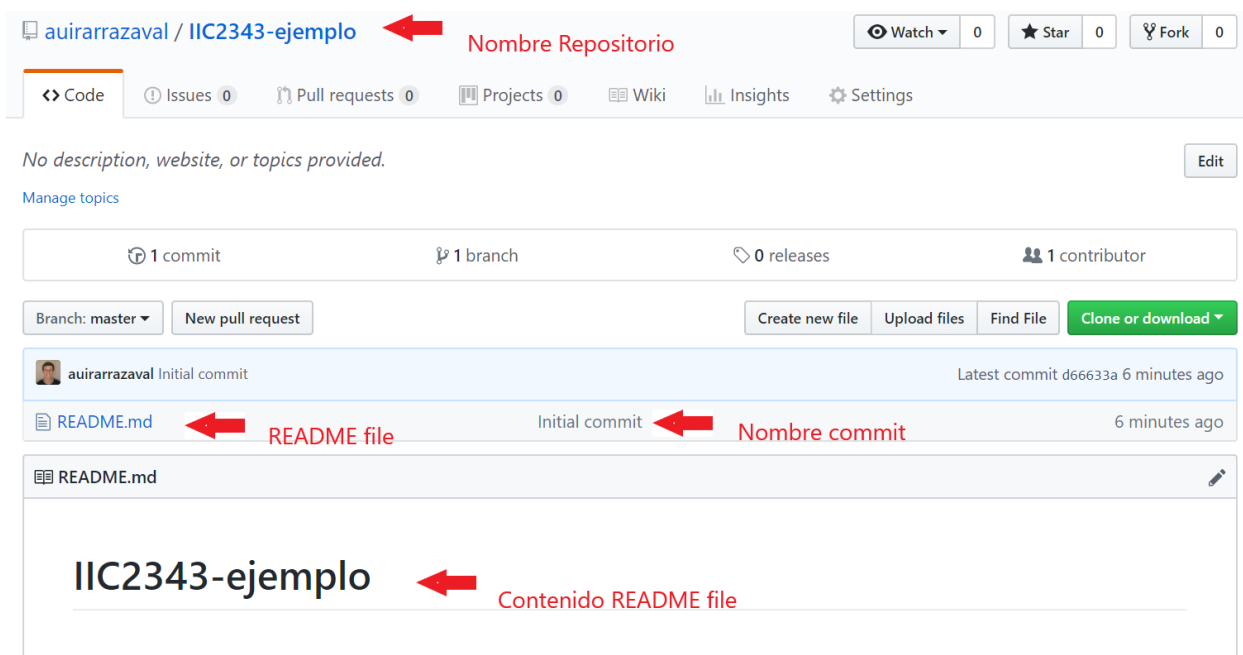
```
Alfonso Irrarázaval :: ~ » git --version
git version 2.20.1.windows.1
```

Una vez ya instalado, podemos empezar a manejar nuestro repositorio de GitHub con Git a través de nuestra terminal, pero antes debemos introducir:

4.2. GitHub & Git

Es una plataforma para alojar proyectos, usando el sistema de control de versiones Git. GitHub nos simplifica la vida entregando una interfaz gráfica donde ver nuestros cambios y distintos estados de nuestro proyecto. Para ejemplificar como funciona GitHub y Git, vamos

a partir con un repositorio vacío, como el que recibirá cada uno al inicio del curso. Al ingresar al link que les entregaremos, se encontraran con algo similar a lo siguiente:



Lo primero que debemos hacer, es **clonar** (crear una versión local de) nuestro repositorio en nuestro computador, para esto, primero debemos ir al botón verde en el repositorio, dónde dice *clone or download*, y se desplegará un menú con el link de nuestro repositorio (el mismo que recibieron por parte nuestra) el cual hay que copiar. Luego deben abrir su terminal, ir a la carpeta dónde quieren que esté su versión local del repositorio y ejecutar el comando:

git clone < link >

En éste caso, quería que el repositorio estuviera en Desktop, por lo que ejecuté *cd Desktop* para ir a esa carpeta (*cd* → Change Directory), y luego ejecuté el comando desde esa carpeta. se ve algo así:

```
Command Prompt
Microsoft Windows [Version 10.0.17134.648]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Alfonso Irarrázaval>cd Desktop

C:\Users\Alfonso Irarrázaval\Desktop>git clone https://github.com/auirarrazaval/IIC2343-ejemplo.git
Cloning into 'IIC2343-ejemplo'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.

C:\Users\Alfonso Irarrázaval\Desktop>
```

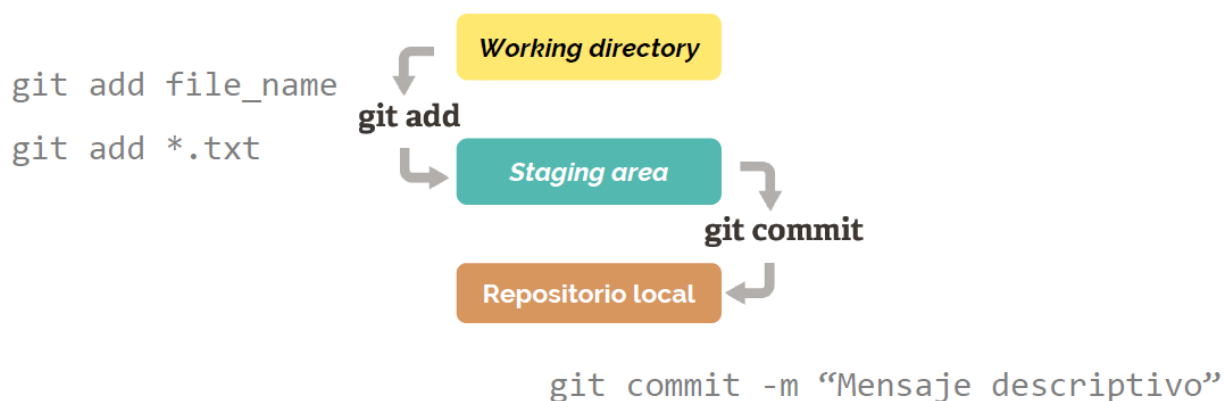
Podemos ver que descargo el repositorio y todos sus contenidos, y ahora existe una carpeta en ese directorio con el nombre de nuestro repositorio, si accedemos a ella (`cd < nombre repositorio >`, o bien a través del explorador) nos podemos dar cuenta que está el **README.md** que tenía en un inicio.

Ahora es necesario hacer una diferencia entre repositorio **local** y uno **remoto**. El local es el que vemos en nuestro computador y editamos directamente. El repositorio remoto corresponde al que está 'online' en GitHub. Lo que necesitamos saber nosotros es, en resumen, como coordinar correctamente estos dos repositorios.

Lo que queremos hacer ahora, es crear un archivo y subirlo a GitHub, para lograr eso crearemos un simple `.txt` (puede ser cualquier tipo: carpetas, `.py`, `.js`, `.vhd`, etc...) al que le pondremos `ejemplo.txt` y escribiremos un texto al azar en él. Este archivo en este minuto se encuentra en nuestra carpeta, pero aún no en GitHub. para agregarlo a GitHub hay que correr los siguientes comandos desde la terminal en el directorio correspondiente:

1. **git add ejemplo.txt** → Agrega nuestro archivo a la **Staging Area**, la cual contiene todos los archivos a los que se les desea hacer *commit*, pueden ser múltiples archivos separados por un espacio, o si se quiere subir todo el directorio, se usa "git add ."
2. (opcional) **git status** → Nos mostrara un resumen de los cambios que se han efectuado en este commit, en este caso, que se agrego el archivo *ejemplo.txt*
3. **git commit -m"nombre commit"** → Este comando se usa para marcar un *checkpoint* en nuestro proyecto, el cual tendrá como nombre/descripción el texto puesto dentro de las doble comillas.

** (Hasta aquí, tenemos agregados los cambios en nuestro directorio **local**, pero si nos fijamos, en GitHub aún no se refleja ningún cambio)



** (Ahora queremos que nuestros cambios se ejecuten en el repositorio **remoto** (GitHub))

4. **git push** → Este comando toma todos los commits pendientes (pueden ser mas de uno) y los “empuja” desde el servidor local al remoto (probablemente pedirá las credenciales username - password de GitHub)

```

C:\Users\Alfonso Irarrázaval\Desktop>cd IIC2343-ejemplo
C:\Users\Alfonso Irarrázaval\Desktop\IIC2343-ejemplo>git add ejemplo.txt
C:\Users\Alfonso Irarrázaval\Desktop\IIC2343-ejemplo>git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   ejemplo.txt

C:\Users\Alfonso Irarrázaval\Desktop\IIC2343-ejemplo>git commit -m"example commit"
[master 9226e39] example commit
 1 file changed, 1 insertion(+)
 create mode 100644 ejemplo.txt

C:\Users\Alfonso Irarrázaval\Desktop\IIC2343-ejemplo>git push
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 321 bytes | 107.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/auirarrazaval/IIC2343-ejemplo.git
   d66633a..9226e39  master -> master
C:\Users\Alfonso Irarrázaval\Desktop\IIC2343-ejemplo>

```

Ahora, si vamos a GitHub, podremos ver que está nuestro archivo ejemplo.txt con su contenido. También podemos ver que sale un “ID” de el commit reciente.

auirarrazaval / IIC2343-ejemplo

Watch 0 Star 0 Fork 0

Code Issues 0 Pull requests 0 Projects 0 Wiki Insights Settings

No description, website, or topics provided. Edit

Manage topics

2 commits 1 branch 0 releases 1 contributor

Branch: master New pull request Create new file Upload files Find File Clone or download

Alfonso Irarrázaval example commit commit ID Latest commit 9226e39 6 minutes ago

File	Commit	Time
README.md	Initial commit	an hour ago
ejemplo.txt	example commit	6 minutes ago

README.md

IIC2343-ejemplo

Si hacemos click en el nombre del commit, podemos ver todas las líneas de los archivos que cambiaron en dicho commit. (Verde es línea agregada, Rojo quitada)

Ya sabemos entonces como llevar nuestro repositorio local al remoto (git push), pero hay casos en los que será necesario llevar contenidos nuevos disponibles en el repositorio remoto a el local (Cuando se suban enunciados en la carpeta Enunciados, por ejemplo). Para esto se hace el opuesto a git push, **git pull**, este comando "descargará" todo nuevo archivo del repositorio remoto al local.

Observaciones Importantes:

1. Si se necesita hacer pull mientras se está trabajando en un proyecto, hay que hacer commit de **todos archivos cambiados** para evitar problemas de consistencia.
2. Al momento de revisar una tarea, se tomará en cuenta el **último commit** antes de la hora de entrega, a no ser que se especifique otro commit en el README

*** Agradecimientos al material de Programación Avanzada 2018-1 por algunas de las imágenes*

4.3. Ejecución en consola

Para lograr el formato de ejecución solicitado deberán utilizar la librería built-in `sys` de siguiente forma:

```
import sys
```

```
### Your Code
```

```
if __name__ == "__main__":  
    path_in = str(sys.argv[1])  
    path_out = str(sys.argv[2])  
    ### Your Code
```

Como pueden ver `sys.argv` corresponde a una lista con los argumentos entregados después de el nombre del programa en la línea de comandos

Política de Integridad Académica

Los alumnos de la Escuela de Ingeniería deben mantener un comportamiento acorde al Código de Honor de la Universidad:

“Como miembro de la comunidad de la Pontificia Universidad Católica de Chile me comprometo a respetar los principios y normativas que la rigen. Asimismo, prometo actuar con rectitud y honestidad en las relaciones con los demás integrantes de la comunidad y en la realización de todo trabajo, particularmente en aquellas actividades vinculadas a la docencia, el aprendizaje y la creación, difusión y transferencia del conocimiento. Además, velaré por la integridad de las personas y cuidaré los bienes de la Universidad.”

En particular, se espera que mantengan altos estándares de honestidad académica. Cualquier acto deshonesto o fraude académico está prohibido; los alumnos que incurran en este tipo de acciones se exponen a un procedimiento sumario. Específicamente, para los cursos del Departamento de Ciencia de la Computación, rige obligatoriamente la siguiente política de integridad académica. Todo trabajo presentado por un alumno (grupo) para los efectos de la evaluación de un curso debe ser hecho individualmente por el alumno (grupo), sin apoyo en material de terceros. Por “trabajo” se entiende en general las interrogaciones escritas, las tareas de programación u otras, los trabajos de laboratorio, los proyectos, el examen, entre otros. Si un alumno (grupo) copia un trabajo, los antecedentes serán enviados a la Dirección de Docencia de la Escuela de Ingeniería para evaluar posteriores sanciones en conjunto con la Universidad, las que pueden incluir reprobación del curso y un procedimiento sumario. Por “copia” se entiende incluir en el trabajo presentado como propio partes hechas por otra persona. Está permitido usar material disponible públicamente, por ejemplo, libros o contenidos tomados de Internet, siempre y cuando se incluya la cita correspondiente.